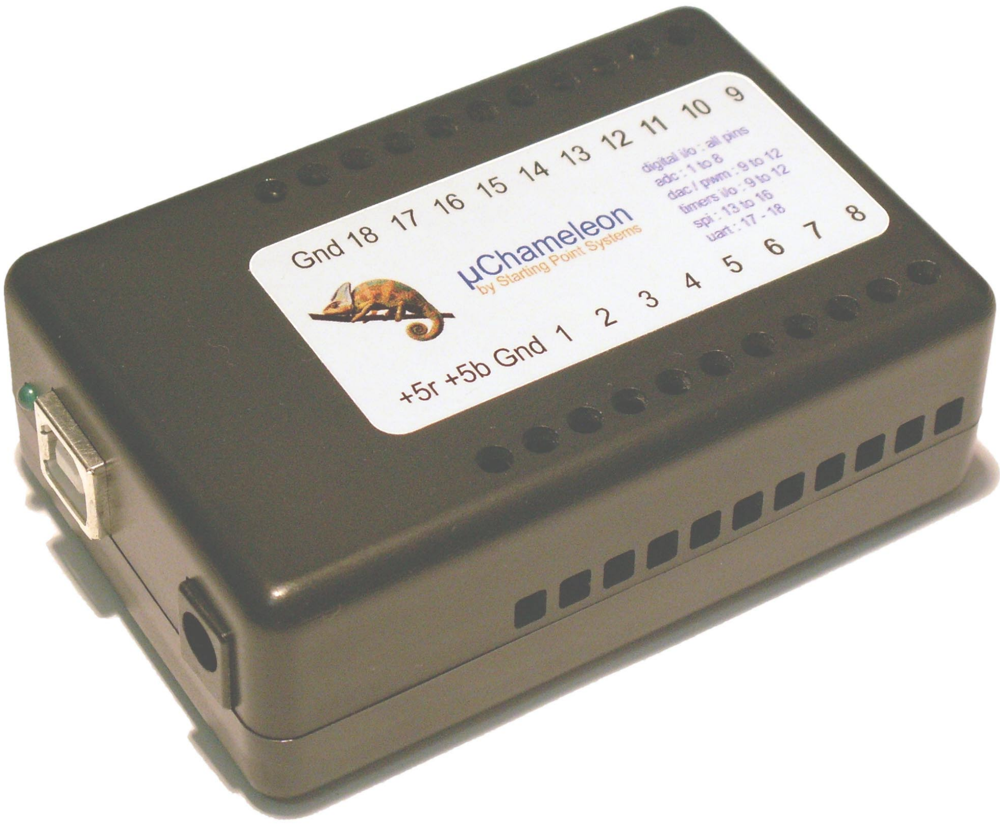
μChameleon User's Manual

Firmware Rev 3.0



1. General overview

1.1. Features summary

- Full-speed USB 2.0 communication bandwidth: 1Mbytes/s
- Drivers available for Windows 98/Me/2000/XP/Vista, Linux, MacOS
- Embedded command interpreter
- Autonomous operation
- 18 general purpose digital I/Os
- 4 timer channels with pulse width modulation
- 8 analog inputs
- Powers from USB, or wall-mount transformer, switches automatically
- Screw connectors to quickly connect your external circuits
- Compact size

1.2. USB communication drivers

There are two main ways to communicate with your µChameleon.
The simplest way is through the Virtual Com Port, or VCP driver interface, where the drivers will emulate a legacy serial port. The port will appear under your "COM and LPT ports" of the control panel. Any existing software or programming environment that can send text strings to a COM port will know how to talk to the µChameleon. You can even perform simple tests with your favourite terminal emulator.
The second way is through a DLL library that replicates the classical Win32 API, and provides for higher bandwidth, and more flexibility, especially if you are planning to manage multiple µChameleons connected to a single computer.

Windows XP, Windows Server 2003 and Windows 2000 drivers support a Combined Driver Model, that allows both alternatives, provided they are not used simultaneously. Windows 98 and Windows Me have one driver that supports the VCP model, and another one that supports the DLL, and will have to be chosen at driver installation.

1.3. Command interpreter

The firmware inside your µChameleon runs a command interpreter that understands full-text commands that let you access all of its hardware resources. The small command set was optimized to provide what is necessary for addressing real-world applications. It hides all the messy details so you can focus on end application, and makes for a short learning curve. No more complicated API that is difficult to learn, with lots of mandatory parameters and subtle and hard to distinguish variants.

1.4. Firmware upgrades

We constantly work to improve the possibilities of the µChameleon, and provide new features following customer input. New firmware can be downloaded from our web site, and with our simple Firmware Upgrader PC software, the newest features are just a click away. Additionally, the hardware protected boot block cannot be erased accidentally, meaning you cannot end up with a locked µChameleon. If something fails during upgrade, simply try again.

1.5. External connectors

The external connectors provide 18 I/O pins. All of them are individually programmable as digital inputs or outputs, and their state can be read or set. Additionally, some of them have specialized functions for use as analog inputs, analog outputs, frequency generation, pulse width modulation…
These special purpose pins are indicated on the label of your µChameleon as a quick reference when connecting to external devices. Here is a summary of these special function pins:

Special function	Applicable pin numbers
digital i/o	all pins
input pull-ups	9 to 16
analog inputs	1 to 8
analog outputs	9 to 12
pwm outputs	9 to 12
timers	9 to 12
spi	13 to 16
uart	17-18

Note: spi and uart are not yet officially supported, please contact us for beta test firmware if necessary.

1. Getting started

1.6. Device driver installation

When you first connect your µChameleon to your computer, you will be prompted for a driver. Please insert the installation disk provided with the µChameleon, and choose the appropriate directory where the driver is located. For example, 'Drivers\Win2k-XP - 2.00.00' if you use Windows 2000 or Windows XP. (See section 1.1)

1.7. Test application: µChameleon control

1.7.1. Overview

On the install cd you received with your µChameleon (also available as a download on our web site) you will find an utility called "µChameleon control", that lets you perform simple tests with a graphical user interface, and that allows you to exert all the features of the µChameleon, as well as show you it is properly connected (if multiple µChameleons are connected to your computer, you will be able to select the one you want to talk to).

Another interesting feature is that for every action you perform in the user interface, the software shows you the command string that is sent to the µChameleon (along with the response if applicable) so you can learn along the process, making for a very short learning curve. In a few minutes, you'll understand how to perform most of the tasks in your own applications.

Additionally, an entry box will let you type random commands that will be sent as is to the µChameleon, again showing the eventual answer.

To install this software, simply launch the "setup.exe" in the "µChameleon control" under "Utilities" directory of the install cd, or grab it on our "downloads" page on our web site.

1.7.2. Automatic device detection

The frame called "Device selection" will show all µChameleon devices connected to your computer, and allow the selection of the device you want to talk to. Of course, if a single device is detected, it will be the default selection.

1.7.3. Activity led toggling

Two buttons, labelled "led on" and "led off" will turn on and off the activity led beside the usb connector of the µChameleon. This is one of the simplest things you can do with it. Additionally, you can use an 8bits word as a sequence for flashing effects, quickly showing a simple state information, without looking at the computer screen. Try typing the following command : "led pattern 5" (and press enter). You should see the led moslty off, with 2 brief flashes. To revert to default state, send: "led pattern 254".

1.7.4. Digital I/Os panel

This panel shows a representation of the µChameleon, where every connector has two clickable items. One is a square box with a letter I or O, and sets the direction on the pin, as input or output, respectively. The other mimics a led, and will show the state of the pin. When programmed as an input, it will be light or dark green, corresponding to the high and low state, respectively. When programmed as an output, the colours will be light and dark red, indicating a high or low output.

1.7.5. Direct command input

This entry box allows you to directly type commands with your keyboard, and send it to your µChameleon. The command is displayed in the logging text box, as well as the answer, if applicable.
Check boxes allow you to select the type of string termination used.

2. Programming reference

This section describes all commands supported by the µChameleon firmware (current version is 2.1). It is possible to try issuing commands, as a learning exercise, by using the "µChameleon control" application. Actions in the graphical interface will also show (in the log text) the command corresponding to each action, and that was actually sent. It is also possible to type commands by hand, in the 'direct command input' box.
Note: In the rest of this manual, when we talk about sending a command, we mean that the corresponding string is sent, followed by a LineFeed, or CarriageReturn, or Cr-Lf combination.

Note: Commands to be sent will be shown in italics, like this: *led on,* as well as replies.

1.8. Communication basics

Depending on your preferences, there are two ways to talk to the µChameleon. The simplest, that will cover most needs, uses the 'Virtual Com Port', that means that everything happens like if you where talking to a device connected to a legacy serial port on your pc. Most programming environments will support that feature, either through built-in functions, or the win32 api. The second way is to use a more direct access, using the provided D2XX dll, which can provide higher throughput and easier multiple-devices support. Of course, it's possible to start writing applications using the COM port interface, and migrate later to the direct interface. Most of the sample code you'll find on our site or this documentation was developed using this approach. Although most programming examples will be provided using 'Visual Basic' style, it is generally straightforward to translate them to other programming languages.

1.8.1. Opening communication

Before actually sending commands, it's necessary to open the communications port, and this will depend on your programming environment, but in Visual Basic, would simply be:

```
MSComm1.PortOpen = True
```

1.8.2. Checking for device presence

Although it is not necessary, you might want to check if your µChameleon is functioning properly and ready to accept commands. It can be done by sending *id*, which the device should respond to by returning *id µChameleon*.

1.9. Activity led

Besides the USB connector of the µChameleon, there is a led that turns on at power-up, with a small off flash, indicating the firmware is up and running, and waiting to receive commands. It is also possible to act on this led by software.

1.9.1. Turning the led on or off

Turing the led on:
led on
led 1

Turn the led off:
led off
led 0

1.9.2. Setting a led flashing pattern

led pattern <n>

The led can be driven by a sequence of 8 'on' and 'off' states, each state corresponding to the state of a bit in the parameter byte. For example, if you want the led to be mostly off, with 3 small flashes, the parameter value can be: 21 (1 + 4 + 16).

Try by sending: *led pattern 21*

1.10. Digital inputs – outputs

1.10.1. Setting pin direction

Setting the n^{th} pin as an input:

pin <n> input
pin <n> in

Setting the n^{th} pin as an output:

pin <n> output
pin <n> out

1.10.2. Reading pin state

Reading the nth pin state:

pin <n> state -> pin <n> 0 | 1

1.10.3. Setting pin state

Setting the nth pin high:

pin <n> high
pin <n> hi

Setting the nth pin low:

pin <n> low
pin <n> lo

1.10.4. Activating pin pull-up

Activating pull-up on pin n:

pin <n> pullup on
pin <n> pullup 1

Deactivating pull-up on pin n:

pin <n> pullup off
pin <n> pullup 0

Note: The pull-up feature is supported only on pins 9 to 16. The typical pull-up resistor value is around 47kOhms.

1.10.5. Monitoring pin activity

This is an alternative method to monitor pin states without having to use a timer in your pc application to periodically poll for the state of pins.

Activating pin monitoring:

pin <n> monitor on | *1*
pin <n> mon on | *1*

Deactivating pin monitoring :

pin <n> monitor off | *0*
pin <n> mon off | *0*

When pin monitoring is active, the µChameleon will constantly check for transitions on the selected pin, and report once per transition by sending the same string that would be returned by the *pin <n> state* command.

For example, if you when to monitor pin 3, you will send:
pin 3 monitor on
and whenever a low to high transition will be detected, you will receive:
pin 3 1
or when a high to low transition happens, you will receive:
pin 3 0

Pin monitoring is supported by all 18 pins, and can be active simultaneously on any pins combination.

1.11. Analog inputs

1.11.1. Reading pin voltage

Read voltage on pin n:

adc <n> -> *adc <n> <v>*

Pins 1 to 8 support the analog to digital conversion of a 0-5volts input to a single byte. The firmware always responds by repeating the *adc <n>,* this allows easy and unambiguous demultiplexing, without waiting for answers each time a command is sent. The <v> value will be in the range [0;255] with 0 for 0volts and 255 for 5volts.

1.12. Analog outputs - PWM - Frequency generation

1.12.1. PWM applications

The four timer channels of your µChameleon can be used to output signals of controlled frequency and duty cycle, and each of them can be used for a variety of purposes, including generation of analog voltages.

A simple R-C filter will generate a clean, linearly varying analog voltage, and by changing this voltage at regular intervals, it is also possible to generate arbitrary waveforms.

This makes it very simple to generate a programmable control voltage, for example, the output voltage of a programmable power supply, the frequency or a vco, in short, anything that can be controlled with an analog voltage.

Is is also possible to use the pwm outputs without any filtering, the pwm output connected directly, for example to control the brightness of a led, of driving a power transistor or bridge, controlling the speed and direction of a dc motor.

1.12.2. PWM commands summary

Here is a summary of the available commands:

pwm <n> on
pwm <n> off
pwm <n> period
pwm <n> width
pwm <n> polarity
pwm <n> prescaler

The following sections give details on using these commands.
Note: these features are available on pins 9-10-11-12.

1.12.3. Pwm channel on or off

Turn on the pwm feature on pin n:

pwm <n> on

Turn off the pwm feature on pin n:

pwm <n> off

Note: these commands can be sent only once at the beginning and end of an application, although it can be a good idea to set various parameters like period and width before turning the pwm on.

1.12.4. Pwm output frequency

Set the signal period of pin n:

pwm <n> period <p>
pwm <n> per <p>

The period parameter is in timer clock cycle units, with a range [0;65535]. Nominal frequency is 10MHz. For example, *pwm 9 period 1000* will program the timer channel on pin 9 for a 10kHz frequency. Note: this is true unless you have prescaled down the clock input – see prescaler section 1.12.7.

1.12.5. Pwm duty cycle

Set the signal duty cycle of pin n:

pwm <n> width <w>
pwm <n> wid <w>

The width parameter is in timer clock cycle units, with a range [0;65535]. For example, if you want to generate a 30% duty cycle signal on pin 9 with a 10kHz frequency, you will send:

```
pwm 9 period 1000
pwm 9 width 300
```

Also, if you want to generate a variable frequency, but with a constant 50% duty cycle, the period being in a variable called myvar, you will send:

```
pwm 9 period myvar
pwm 9 width myvar/2
```

Note: this syntax is a simplification, because myvar and myvar/2 should be sent as string, and because other commands might be necessary, notably turning the pwm channel on if this is the first action.

1.12.6. Pwm polarity

It is possible to control the polarity of the logic level of a pwm channel, which will affect the meaning of the width parameter (a 30% duty cycle would now become 70%).

Normal polarity: (default)

```
pwm <n> polarity 0
pwm <n> pol 0
```

Inverted polarity:

```
pwm <n> polarity 1
pwm <n> pol 1
```

Note: pwm channels 9 and 10, as well as 11 and 12 share their clock, so you can easily generate complementary signals by programming two channels identically, and just inverting the polarity of one of then.

1.12.7. Pwm prescaler

The period and width parameters of the pwm feature are expressed in terms of cycle counts, with a default frequency of 10MHz, that is a resolution of 100ns. This means that the minimum possible frequency is about 153Hz. It is also possible to prescale the clock of pwm channels by the following amounts: 1,2,4,8,16,32,64. Default: 1.

pwm <n> prescaler <p>
pwm <n> pre <p>

1.13. Variables and arithmetic

1.13.1. Dim

The dim statement create a new variable, stores its name and type, and allocates memory for it. The variable name and type are stored permanently in the flash memory, but the value is not retained when power is lost.

Syntax: *dim <varname> as <type>*

Example: *dim myvar as int*

Notes: It is possible to define up to 32 variables. Variable names may have up to 12 characters. Only the 'int' type (16 bit integer) is defined in firmware 3.0.

1.13.2. Let

The let statement assigns a value to a variable, the right side of the equal sign being either a single variable or constant, or an arithmetic operation combining two variables or constants.

Syntax: *let <variable> = <value>*
let <variable> = <value> <operator> <value>

Examples: *let x = 0*
let counter = counter + 3
*let speed = speed * accel*

Available operators:

operator	arithmetic performed
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

1.13.3. Increment and decrement

Instead of writing:	*let var = var + 1*
One can write:	*increment var*
Or:	*incr var*

Instead of writing:	*let var = var - 1*
One can write:	*decrement var*
Or:	*decr var*

This is generally more convenient, results in more compact code, and is also faster.

1.13.4. Print

It is possible to query the value of a variable with the print statement.

Syntax: *print <varname>*

This command will send the variable content to the USB communication port.

1.13.5. The '?' special variable

Some former commands, like 'adc' or 'pin', have their last value stored in a special variable that can be accessed with the '?' sign.

Example: *adc 1*
let voltage = ?

the 'voltage' variable (that we suppose was created previously with 'dim') now contains the result of the last analog to digital conversion.

Note: the '?' sign can be used in any place a variable or constant would be used.

1.13.6. Erasing variables

It is possible to erase all variables, and their definitions, as well as freeing the memory they use with the erase command:

Syntax: *erase dims*

1.14. Conditional execution

1.14.1. If … then and relational operators

If statements provide the ability to conditionally execute commands based on comparison between values.

The general form it takes is the following:

```
if <value A> <operator> <value B> then <command>
```

The value parameters can be any variable or constant. The operator can be any of the following:

operator	comparison performed
=	equal to
<>	not equal to
>	strictly greater than
>=	greater or equal
<	strictly less than
<=	less or equal

Examples:

if counter >= 1024 then let counter = 0
if voltage < 128 then pin 3 low
if alert = 1 then led pattern 5

1.15. Event handlers

1.15.1. Introduction

Event handlers provide a way to store a list of instructions to be executed when a specific event occurs. They are similar to functions or procedures in most programming languages except they don't take parameters.

There are currently two defined events: 'reset' and 'background'.

1.15.2. The reset event

As the name implies, the reset event occurs once at power-up and every time the µChameleon is reset, either via software, by pressing the reset button, or when power is cycled.

1.15.3. The background event

The background event is a periodic event triggered by an internal oscillator running at approximately 20Hz. It can be turned on or off (default after reset is off).

Syntax for turning on the periodic event generator:

background on
back on

Syntax for turning off the periodic event generator:

background off
back off

1.15.4. Defining event handlers

Syntax for defining event handlers:

```
onevent <event name>
<instruction line 1>
<instruction line 2>
.
.
.
<instruction line n>
endevent
```

Example:

```
onevent reset
pin 2 output
background on
endevent

onevent background
adc 1
if ? > 135 then pin 2 low
if ? < 126 then pin 2 high
endevent
```

Note: The preceding example show how simple it is to implement a simple standalone temperature controller with hysteresis.

2. Hardware Information

2.1. *Inputs / Outputs*

All inputs are very high impedance CMOS inputs (typically greater than 10Mohms.)
All I/Os are protected with 100Ohms current limiting resistors. This enables a direct connection to LEDs, opto-couplers, power transistors, miniature relays, piezo buzzers, small loudspeakers… with a typical current output of 20mA.

2.2. *Power supply circuitry*

2.2.1. Power circuitry overview

- Powers from USB
- Powers from wall-mount transformer
- Powers from local regulated +5 Volts
- Switches automatically between its power sources
- Provides power to your external circuitry
- Multiple protection schemes ensure high reliability

The power circuitry of the µChameleon is extremely flexible, and has been designed to adapt to as many real-world situations as possible.
First, it can be powered directly by the USB port of your computer, which is the default configuration most users are satisfied with.
In some instances however, for example when using an bus powered hub that is not capable of providing enough current, or to conserve the battery of a laptop, or to get an accurate 5 Volts level for some analog applications, the µChameleon has an internal linear regulator.

2.2.2. Power circuitry protections

The power circuitry is protected against the following situations:

- Polarity inversion of wall-mount connection
- Board power short circuit
- Current consumption over 500mA (protects your computer)
- Over-temperature of power switch
- Over-temperature of linear 5Volts regulator

2.2.3. Using an external wall-mount transformer

Simply connect a standard wall-mount transformer, with a typical output voltage between 9Volts and 12Volts to the black connector beside the usb connector. The external voltage will be internally regulated to a clean 5Volts, and the µChameleon will switch from USB power automatically.

2.2.4. Thermal considerations

When powered by an external transformer, the region near the power connector can feel warm to the touch. This is normal. However, it is suggested not to apply more than 16 Volts to the external power input, especially at high currents, as this will increase the heat produced by the internal linear regulator.